BLM3011 - Operating Systems
Lecture Notes

Şafak Bilici

2020-2021

What Is An Operating System?

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and a user.

- Hardware – provides basic computing resources
  - CPU, memory, I/O Devices
- Operating System
  - Controls and coordinates use of hardware among various applications and users.
- Application programs – define the ways in which the system resources are used to solve the computing problems of the users
  - Text editors, compilers, web browsers, database systems, video games
- Users
  - People, machines, other computers

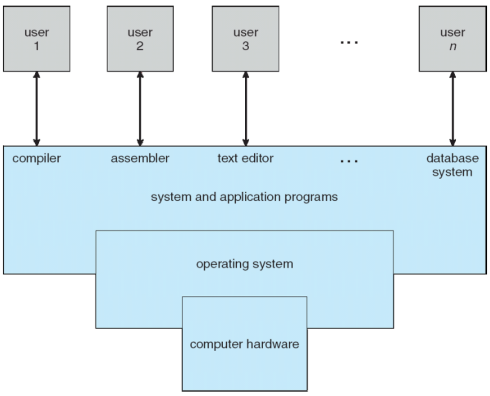


Figure 1: Computer System

A program that acts as an intermediary between a user of a computer and the computer hardware. Major functions of operating systems **may** include:

- Managing memory and other system resources.
- Imposing security and access policies.
- Scheduling and multiplexing processes and threads.
- Launching and closing user programs dynamically.
- Providing a basic user interface and application programmer interface.

Not all operating systems provide all of these functions. Single-tasking systems like MS-DOS would not schedule processes, while embedded systems like eCOS may not have a user interface, or may work with a static set of user programs.

An operating system is **not**

- The computer hardware.
- A specific application such as a word processor, web browser or game.
- A suite of utilities (like the GNU tools, which are used in many Unix-derived systems).
- A development environment (though some OSes, such as UCSD Pascal or Smalltalk-80, incorporate an interpreter and IDE).
- A Graphical User interface (though many modern operating systems incorporate a GUI as part of the OS).

While most operating systems are distributed with such tools, they are not themselves a necessary part of the OS. Some operating systems, such as Linux, may come in several different packaged forms, called distributions, which may have different suites of applications and utilities, and may organize some aspects of the system differently. Nonetheless, they are all versions of the same basic OS, and should not be considered to be separate types of operating systems.

What Operating System Do?

It depends on the point of view.

- Users want convenience, *ease of use* and *good performance*
    - Don't care about *resource utilization*
- But shared computer such as mainframe or minicomputer must keep all users happy
- Users of dedicate systems such as workstations have dedicated resources but frequently use shared resources from servers
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

OS is a *resource allocator*. It manages all resources and decides between conflcting requests for efficient and fair resource use. Also OS is an *control program*. It controls execution of programs to prevent errors and improrer use of the computer.

The "one program running at all times on the computer" is the *kernel*. The kernel of an operating system is something you will never see. It basically enables any other programs to execute. It handles events generated by hardware (called interrupts) and software (called system calls), and manages access to resources. The kernel usually defines a few abstractions like files, processes, sockets, directories, etc. which correspond to an internal state it remembers about last operations, so that a program may issue a session of operation more efficiently.

Computer Startup

A *bootstrap program* is loaded at power-up or reboot. Typically stored in ROM or EPROM, generally known as *firmware*. It initializes all the aspects of the system, loads operating system kernel and starts execution.

Computer-System Organization

A modern general-purpose computer system consists of one or more CPU s and a number of device controllers connected through a common *bus* that provides access between components and shared memory.

Each device controller is in charge of a specific type of device (for example, a disk drive, audio device, or graphics display). Depending on the controller, more than one device may be attached.

A device controller maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. CPU moves data from/to main memory to/from local buffers.

Typically, operating systems have a *device driver* for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device. The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

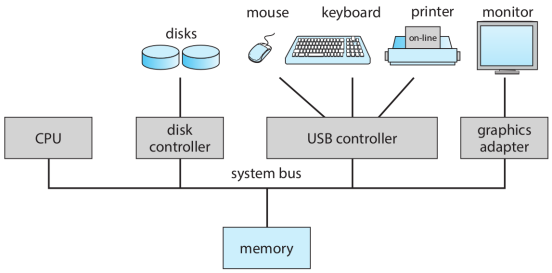


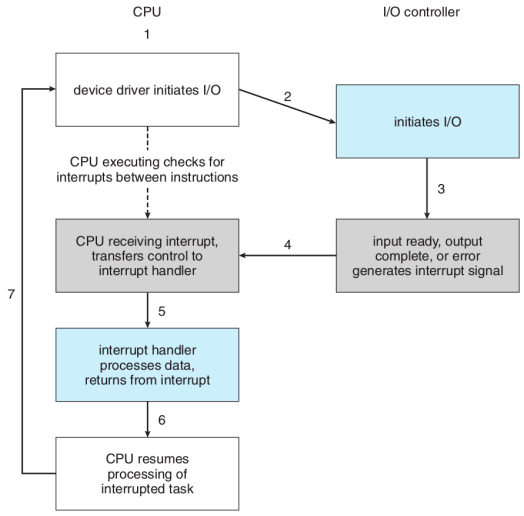
Figure 2: A typical PC computer system.

Interrupts

Consider a typical computer operation: a program performing I/O . To start an I/O operation, the device driver loads the appropriate registers in the device controller. The device controller, in turn, examines the contents of these registers to determine what action to take (such as "read a character from the keyboard"). The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver that it has finished its operation. The device driver then gives control to other parts of the operating system, possibly returning the data or a pointer to the data if the operation was a read. For other operations, the device driver returns status information such as "write completed successfully" or "device busy". But how does the controller inform the device driver that it has finished its operation? This is accomplished via an **interrupt**.

Hardware may trigger an interrupt at any time by sending a signal to the CPU , usually by way of the system bus. Interrupts are used for many other purposes as well and are a key part of how operating systems and hardware interact.

Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines. Interrupt architecture must save the address of the interrupted instruction. A *trap* or *exception* is a software-generated interrupt caused either by an error or a user request. An operating system is *interrupt driven*.



(a) Interrupt-driven I/O cycle.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

(b) Intel processor event-vector table.

Figure 3

The CPU hardware has a wire called the *interrupt-request* line that the CPU senses after executing every instruction. When the CPU detects that a controller has asserted a signal on the interrupt-request line, it reads the interrupt number and jumps to the *interrupt-handler routine* by using that interrupt number as an index into the interrupt vector. It then starts execution at the address associated with that index.

We say that the device controller **raises** an interrupt by asserting a signal on the interrupt request line, the CPU **catches** the interrupt and **dispatches** it to the interrupt handler, and the handler **clears** the interrupt by servicing the device.